SQL Rules of Thumb

Native Coding

The following are some design tips for coding SQL statements. The following example will be used to illustrate some of the following points:

Table1 C1, C2, Gender,BeginDate, EndDate (C1 is the primary key)
Table2 C1, C2, FKC1 (C1 is the primary key, FKC1 is a foreign key to Table1)

Performance

1. When joining tables, **always** specify the join condition first **using only SQL variables**. This helps ensure that the tables will be accessed properly.

   Correct:

```
SELECT A.C1
      , B.C2
  FROM Table1 A
      , Table2 B
WHERE A.C1 = B.FKC1
  AND A.C1 = :DG998-C1
```

   Incorrect:

```
SELECT A.C1
      , B.C2
  FROM Table1 A
      , Table2 B
WHERE A.C1 = :DG998-C1 (No join conditions -
only host variables)
  AND B.FKC1 = :DG999-C1
```

   To ensure the developer does not forget to add adequate join conditions, it would be preferable to use the new join syntax available in DB2 V4.1 or later:
   *

   Correct:

```
SELECT A.C1
      , B.C2
  FROM Table1 A INNER JOIN Table2 B
```

```
ON A.C1 = B.FKC1
WHERE A.C1 = :DG998-C1
```

- If you have additional host variables which may be specified on primary key columns, they should be specified next.

- Next, you should specify predicates in most discriminating order. This simply means that you should specify selection criteria which will return the least amount of rows next. If you were to code two additional predicates, say, Gender, and Birthday, you would specify Birthdate first, and then Gender. There are only two possible values for Gender (presumably) - Male and Female. There are many possible values for Birthdate (out of the many people you know - who has the same birthdate?).

2. Only select what you need; you do not need to SELECT columns just because they are in your WHERE clause (but they are required if you are specifying the column in the ORDER BY clause).

   The idea here, is that you are consuming excess system resources. The more you return to your program, the more that has to cross system address spaces. This translates into increased memory usage and CPU consumption.

   - This is especially true when using intersection entities (also known as an associative entity type). In #1, notice that B.FKC1 wasn't selected. Since, by definition of the equality on the join condition, both A.C1 and B.FKC1 will contain the same value. Likewise, there is no need to return A.C1 since it's value must be equal to host variable DG999-C1.
   - If you can limit the selected columns to those columns that are contained within one or more indices, DB2 will avoid retrieving the data pages altogether which results in significant I/O savings.

3. **Always** use the correct host variable type to hold a given SQL variable! If you are not sure what its type should be, then check an existing DCLGEN.

   Due to the underlying storage of certain variable types, the use of a "less optimal" data type could result in extremely long execution times. Any time that DB2 must convert data in a host variable, that variable becomes ineligible for index processing. I have seen cases where program execution time has gone from one hour to five minutes simply by correcting a host variable definition.

4. If you are checking for an **inclusive** range BETWEEN is more efficient than >= and <=. Note: This is for an inclusive range!

Correct:

```
SELECT A.C1
    FROM Table1 A
  WHERE CURRENT DATE BETWEEN BeginDate
and EndDate
```

Incorrect:

```
SELECT A.C1
    FROM Table1 A
  WHERE CURRENT DATE >= EndDate
    AND CURRENT DATE <= BeginDate
```

The BETWEEN clause also aids in statement readability.

5. The IN clause is more efficient than OR clauses for checking a list of values; this also aids in statement readability.

Correct:

```
SELECT A.C1
    FROM Table1 A
WHERE A.C1 IN (1, 2, 3, 4)
```

Incorrect:

```
SELECT A.C1
    FROM Table1 A
  WHERE A.C1 = 1
      OR A.C1 = 2
      OR A.C1 = 3
      OR A.C1 = 4
```

6. Do not use the ORDER BY clause unless it is actually needed. If order is important, then you should specify it; otherwise the order will not be guaranteed (e.g. do not depend on a clustering index to guarantee order, because the access path or the attributes of the index may change.).

Unless you are strictly selecting indexed columns, DB2 will have to perform a sort which is an expensive operation. Take a moment to consider your application - if you are writing a file which will later be sorted, don’t use the ORDER BY clause. Again, there are situations where its use is justified. An ORDER BY is one of the most "expensive" operations you can perform in a SQL statement.

7. If possible, ORDER BY statements should only contain columns from the same table.

   If the ORDER BY clause contains columns from more than one table, DB2 will always perform a sort.

   Correct:

```
SELECT B.FKC1
     , B.C2
  FROM Table1 A
     , Table2 B
WHERE A.C1 = B.FKC1
  AND A.C1 = :DG998-C1
ORDER BY B.FKC1, B.C2
```

   Incorrect:

```
SELECT A.C1 (redundant column being selected)
     , B.C2
  FROM Table1 A
     , Table2 B
WHERE A.C1 = B.FKC1
  AND A.C1 = :DG998-C1
ORDER BY A.C1, B.C2 (ordering by columns of two
```

   different tables)

8. If you can, convert negative statements to positive statements (e.g. use IN as opposed to NOT IN). A NOT will convert a Stage 1 predicate to a Stage 2 predicate resulting in degraded performance (If you really want to know what a Stage 1 or Stage 2 predicate is, see your DBA and they will be happy to explain it to you, but bring a pillow). NOT logic is typically more difficult to read.
9. **DB2 joins are significantly more efficient than an application join! You should assume that DB2 can perform the join more efficiently unless you can disprove it. Coding SQL joins maintains the additional benefits of simplifying application code and having the opportunity to take advantage of future SQL enhancements in DB2.**
10. Add COMMIT processing and restartability to your application to minimize the time required to restart the application in the event it abnormally terminates due to programming or external processing errors. An application should typically COMMIT every 30 seconds.
    If an application abnormally terminates all work will be rolled back to the last commit point. If the application is not restartable, processor resources will be wasted (due to the duplicate amount of work which must be performed). COMMITs also free locks held on DB2 resources allowing greater concurrency. COMMITs should typically be issed at 30 second intervals (again, to free

resources and allow greater concurrency); DB2 will wait up to 60 seconds for a resource to be freed before issuing a deadlock and terminating the application thread.

11. If you will be processing across multiple units of work, you should declare all cursors with the WITH HOLD option.
    If you are issuing COMMITs in your application without declaring cursors with the WITH HOLD clause (and there is the pontential that the restart package may issue COMMITs on your behalf), all cursors will be automatically closed by DB2. This would require the application to perform "re-positioning" logic. The WITH HOLD clause prevents the cursors from being closed at commit time.
12. Avoid the use of arithmetic within SQL statements.*
    Arithmetic in a SQL statement will cause DB2 to avoid the use of an index. The cost of the additional I/O operations required to complete the same query far outweighs simplicity of coding. The arithmetic should be moved to the application code (e.g. COBOL COMPUTE instruction) where it may be performed more efficiently.
13. Use DISTINCT only when necessary.
    The use of DISTINCT will always cause DB2 to perform a sort – even if it is impossible to for the result set to contain duplicates.

General

1. The DB2 optimizer is fairly sophisticated, but you should provide as much information as you can give it; this can be translated as "provide as much selection criteria on your WHERE clause as is known to your program at the time the SQL call is invoked". You should also attempt to use SQL variables to reduce the amount of rows returned to your application program.
   This gives DB2 more information to use to determine the optimal data path. DB2 can eliminate rows more efficiently than your application can, so you should use this feature to your advantage.

Style

The following are suggestions only, but you should strive to provide adequate indentation and whitespace to make your SQL statement as readable as possible. If you use a consistent style, it will also aid other developers reading your code. Look at the various examples above and you will notice the following:

1. SELECT, FROM, WHERE or other reserved words should be right-aligned against the SELECT statement.
2. Operators are aligned with ample white space between their variables. This allows developers to scan down the line for specific conditions.
3. Tables are listed one per line.
4. Tables are qualified with a correlation variable (e.g. A and B in most of the above examples). These should be short and should help identify the original table. This isn’t always required, but you should use them if you are joining several tables.

5. Commas (to continue a clause) are at the beginning of the next line.
6. Although I don't show it in the examples above, always add a comment containing **one** English **grammatically correct** sentence (and only one sentence) to describe the SQL statement. This will aid developers who must maintain the code and will immediately give them an idea of the nuances of the code (e.g. some aspects of a SQL statement are implied rather than directly observable). An example may be "This cursor will retrieve all ORGs that don't have any cases associated with them".
7. You should always provide a comment when special "tricks" are used to influence the optimizer. This is critical because these are typically DB2 version-specific predicates and often may confuse other developers maintaining the code. The comment should include what statement was added for performance optimization, and the intended effect of the statement.

COBOL Coding

The DB2 Application Programming and SQL Guide contains information related to language-specific implementations of DB2 Embedded SQL.

1. **Never** code "SELECT *" in an embedded SQL statement.
   When a "SELECT *" is embedded in an application program and the program is compiled, the DB2-precompiler establishes a fixed set of working storage variables (memory) to contain the results of the SQL statement. If the table is altered to add new columns, any subsequent execution of the embedded SQL statement will include the additional column; this implies that the program did not previously acquire enough memory to hold the data returned by DB2 (which would lead to a Operation Exception abend, or corruption of program working storage).
2. Always code a column list on an INSERT statement.
   The idea behind this is similar to the "SELECT *" rule. When an INSERT statement is coded wihtout a column list, DB2 assumes that a host variable is present for every column in the table. If an additional column is added to the table, the program will have to be changed to include a new host variable. If the column list is present, DB2 assumes that any columns that are absent from the list are either NULL, or NOT NULL WITH DEFAULT, so the program does not have to be changed to populate the new column.
3. Always provide a null indicator for nullable columns, and check its value prior to using its equivalent host variable value; this is also true of functions applied to non-nullable columns, but which may still return a NULL value. If you do not wish to add additional logic required to deal with NULL values, you should apply the COASLESCE or VALUE functions to the nullable column.
4. Nullable columns have the option of containing "nothing". DB2 does not initialize these fields and may contain invalid data (e.g. alphabetic data in numeric fields). To aid in source code readability, COBOL 88-levels (or equivalents for other languages) should be defined in the null-DCLGEN.

Miscellaneous Definitions

Stage 1 Predicate A Stage 1 predicate is evaluated at the lowest address space within DB2. It is preferable that a predicate is a Stage 1 predicate because it means that rows are eliminated as soon as possible and are eliminated from subsequent processing such as sorting as required to satisfy an ORDER BY clause.

Stage 2 Predicate A Stage 2 predicate is applied to remaining rows after all Stage 1 predicates have been satisfied (and rows not meeting this criteria are eliminated).

---

*This is DB2 version-specific item and is subject to change in subsequent DB2 releases.